Lazy Evaluation

* Thunks are fast if you never need the computation, but slow if you need the computation a lot
* Lazy Evaluation lets you avoid computation until needed, but remember it for later

Delay and Force

* Delay evaluation of X by storing it in a promise
* A promise is a 2-element mutable list
* Place false in the head, and thunked X in the tail
* Force X, by set the head of the promise to true and the tail to the evaluation of X

Stream

* An infinite sequence of values
* Thunks that return a pair (value, stream)

Memoization

* If a function has no side effects and does not read mutable memory
  + Keep a cache of previous results
  + Benefits realized if maintaining cache is cheaper than recomputing and if cached results are reused
* Similar to promises, but if the function takes arguments, then there are multiple “previous results”
* For recursive functions, this clan lead to exponentially faster programs (this is related to the technique dynamic programming)

Macros

* Macro definition describes how to transform some new syntax into different syntax into different syntax in the source language
* Macro system is a language for defining macros
* Macro expansion is the process of rewriting syntax for each macro use before the program runs

Tokenization

* Macro systems work at the level of tokens, rather than sequences of characters
* “macro expand head to car” doesn’t rewrite “head-door” to “car-door”

Paranthesization

* Macro use: (macro-name …)
* After expansion: (evaluated-macro)